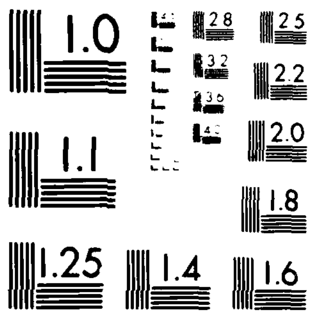
AD-A122 464 COMPARISON OF MICROPROCESSOR BASED DATA BASE MANAGEMENT 1/1
SYSTEMS(U) NAVAL POSTGRADUATE SCHOOL MONTEREY CA
R B RELUE JUN 82

UNCLASSIFIED F/G 9/2 NL

END
DATE
FILMED
1-83
DTIC



MICROCOPY RESOLUTION TEST CHART
NATIONAL BUREAU OF STANDARDS 1963-A

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

COMPARISON OF MICROPROCESSOR BASED
DATA BASE MANAGEMENT SYSTEMS

by

Richard B. Relue

June 1982

Thesis Advisor: Dushan Badal

Approved for public release; distribution unlimited

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A122164	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Comparison of Microprocessor Based Data Base Management Systems		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis June 1982
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Richard B. Relue		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		12. REPORT DATE June 1982
		13. NUMBER OF PAGES 91
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office)		15. SECURITY CLASS. (of this report)
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release, distribution unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

DBMS

Database

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

In this report two leading DBMS designs (DBASE II and MDBS III) are evaluated in terms of their design and performance on Altos computer systems. In addition some comments are made on their relative advantages and disadvantages for the average user.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

The basic conclusions reached are that the relational design of DBASE II is easier to use and apparently performs data manipulation problems better than MDBS III. In large scale applications where reliability and security are imperative, MDBS III is more appropriate since it provides crash recovery, access control, integrity constraints and other key features which DBASE II lacks.

Accession For

NTIS GRA&I X

A

Approved for public release, distribution unlimited

Evaluation and Comparison of Microprocessor Based Data Base Management Systems

by

Richard B. Relue
Lieutenant Commander, United States Navy
B.S., Florida Institute of Technology, 1972

Submitted in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
JUNE 1982

Author: Richard Relue

Approved by: Thesis Advisor

Second Reader

Chairman, Department of Computer Science

Dean of Information and Policy Sciences

ABSTRACT

In this report two leading DBMS designs (DBASE II and MDBS III) are evaluated in terms of their design and performance on Altos computer systems. In addition some comments are made on their relative advantages and disadvantages for the average user.

The basic conclusions reached are that the relational design of DBASE II is easier to use and apparently performs data manipulation problems better than MDBS III. In large scale applications where reliability and security are imperative, MDBS III is more appropriate since it provides crash recovery, access control, integrity constraints and other key features which DBASE II lacks.

TABLE OF CONTENTS

ACKNOWLEDGEMENT

Some of the information presented in this thesis references names which are trademarks. Rather than list each reference separately they are listed here.

CPM-86 and PL/I-80 are trademarks of Digital Research

CP/M, MP/M and CP/NET are trademarks of Digital Research

DBASE II is a trademark of Ashton-Tate

MDBS III is a trademark of Micro Data Base Systems, Inc.

I. INTRODUCTION

The use of Database Management Systems (DBMS) in the microcomputer environment is relatively new development. Until recently there were no fully implemented DBMS commercially available for microprocessor based computer systems. Commercial distributors have now begun to offer several DBMS which have the capabilties previously available only on main-frame computer and minicomputer systems.

While large computers are already using many successful applications packages developed in conjunction with commercial Database Management Systems, DBMS packages for large systems are too costly and require too much memory to be adapted to small systems. Consequently new DBMS packages had to be developed from the beginning so that they worked within the limitations and capabilities of microcomputer systems.

Despite the fact that these new systems were developed independently of main-frame DBMS systems they still retained many of the sophisticated features and problems associated with these earlier large machine based DBMS systems. Some of these problems are:

1. Concurrency control
2. Security
3. Crash protection and recovery
4. Data model selection
5. Operating system dependency

6. Transportability

7. User interface and host language

8. Distributed and Network DBMS design

9. Consistency constraints

10. Performance

In addition to the above problems, the microprocessor based systems have very real hardware limitations (ie. limited memory and speed). Any practical design would have to minimize its use of memory and computing time.

The purpose of this project was to do a comparative study and evaluation of these microprocessor based designs. This in essence meant determining what these designs were capable of and what methods were used to implement them. In addition we explored the some of the common problems encountered in DBMS design and how they are handled.

For purposes of the study, two DBMS systems were selected to illustrate what is possible with small microprocessor systems and to evaluate their relative advantages and disadvantages. The two systems selected were DBASE II and MDBS III. DBASE II is a relational DBMS which interprets its commands at run time. MDBS III is a network DBMS which uses a host language and is compiled prior to execution. These two systems were tested on the CP/M and MP/M operating systems to get a comparison of their relative speed in performing similar problems.

II. DATABASE MANAGEMENT SYSTEM CONCEPTS

When computers first became available commerciall, they were applied to mathematical problems, but it soon became evident that computers were also useful for storage and retrieval of information. Business systems were developed to take advantage of the computers ability to rapidly access and manipulate data. These early systems were essentially file management systems. They required specific programs to enter and retrieve data from the database. In addition, the structure of the database was usually limited by what data structures and capabilities were available in the selected language. Many of them were written using commonly available business languages such as Cobol.

As the databases became larger and more complicated, modification of software for the database and efficient storage of information became more difficult. It became evident that a system was needed which would automatically organize a database to user specifications and provide the user with a simple logical inter ace to the database. These systems were referred to as DATABASE MANAGEMENT SYSTEMS (DBMS).

The purpose of the DBMS was to allow the user to see the database in abstract terms. The user would not have to know exactly where the data was and in what form it was physically stored. In addition, the user was not concerned about the actual retrieval and storage of data since this also was taken care of by the DBMS. The DBMS visible to the user is essentially a high

level language geared to data storage and retrieval. The user saw the data only as logical structures which he created in this high level language.

The computer only understands binary numbers and thus deals with the database in those terms. The user, on the other hand only sees abstractions. For instance a bank teller looking at a database of checking accounts would see account numbers, withdrawals, deposits and the name of the owner of the account. This difference is generally referred to as different levels of abstraction.

These levels of abstraction define the architecture of a DBMS. In defining the general architecture of a DBMS we can usually divide it into four levels. At the outermost level is the user and his workspace. The next level is the user interface to the database. Next comes the conceptual database which defines the logical scheme or design of the data structures used in the database to store information. At the lowest level of abstraction is the physical database which in its simplest form is binary data stored on a disk.

The user of the database sees a "view" of the database which is limited by the program (ie. application) he is using. Note that he does not have to understand much about the whole database. The program can display messages to the user which will explain all he needs to know to use the program.

Below this user level, is the user interface to the DBMS and the conceptual database. A programmer at this level would see

logical data structures containing character strings and numbers. The logical organization of the conceptual database is referred to as the "conceptual schema".

There are three generally accepted conceptual schema or data models for a database. They are as follows:

1. Hierarchical - The data model for this is a tree. The tree starts with single node which is divided into subnodes. These subnodes are in turn subdivided. The links between these nodes are regarded as relationships. Consider the case of an employer and his employees. One node may represent data about an employer while the subnodes represent data about his employees.

2. Network - This is similar to the hierarchical except that the links do not have to be arranged in the form of a tree. For a network the correct data model is a graph. The nodes represent records which have relationships defined by the links between the nodes. This system can represent very complex relationships and more nearly represents real world situations.

3. Relational - This model divides the database into a series of data tables which specify relationships between data items. The presence of different items in the same record of this table implies a relationship between them.

Below the conceptual database is the physical database with its "physical schema". It is possible to view the physical database a simply a collection of binary bits. Usually though it is as viewed as a collection of simple data structures or files stored in secondary media.

This chain of abstractions from the user to the conceptual database to the physical database is important in database design since it provides what is called "data independence". The physical schema can be modified (ie. new disk storage system) without changing the conceptual schema and applications programs which use it. This is referred to as "physical data independence". The conceptual schema can be changed without modifying the host language interface and the application programs which use it. This is referred to as "logical data independence". This data independence is important since it makes DBMS design more portable and less dependent on the particular hardware system upon which it is implemented.

As previously mentioned, the user program or application is the highest level of abstraction in the DBMS. This program is written in terms of some high level language called a "host" language. The host language must have some interface to the DBMS conceptual database.

Usually three different interfaces are defined for the the conceptual database. One is called the Data Definition Language (DDL) which is used to specify the conceptual structure of the database. Usually it also allows the user limited control over the physical database which is implemented from the conceptual database.

The next interface is the Data Manipulation Language (DML). This is used to insert, delete, modify and find data in the database. Although the language terminology is aimed at the

conceptual database, in reality it acts upon the physical database.

The third interface is the Query Language (QRS). This allows the user to extract information from the database by locating records that the user desires. This is useful for asking questions about the data and summarizing it.

In addition to the traditional functions of information storage and retrieval a DBMS must be able to handle some special problems. These problems are related to the multiple user environment and reliability of the database.

Databasescan become very large and their continuous availabiltity can become very important to a user. To an airline the loss of their database even for a few hours could be disastrous financially. To prevent these problems a database must have crash protection and recovery. What this means is that the DBMS must be able to protect itself in case of a hardware or media failure. If the database is damaged there must be some means to reconstruct the database to make it current again.

With multiple users accessing the file we have concurrency and synchronization problems. This problem occurs when two or more users are accessing the same database. If one user reads a record and then modifies it there is no problem. Suppose though, that two users simultaneously read a record and modify it. The final version of the record will depend on what order the users of the database read and wrote the record. The DBMS must have a

scheme to make the process of modifying a record independent of the order in which users access it.

Another problem related to multiple users is protecting the database from unintentional or malicious damage. A database may contain many different but related items. Many different users may require access to the information. In such an environment it is easy for users to delete records by mistake or make erroneous changes to it. The ability of the database to protect itself from unintentional errors and malicious attempts to destroy it is referred to as the "integrity" of the database. Enforcement of range limits on the addition of new data and modification of old data is referred to as "consistency constraints".

When a database has many different users, usually most of them do not need access to all the data. Indeed it may be important to prevent unauthorized access since some of this information may be of a private or classified nature. Some example of this are pay records, trade secrets and classified military information. This is referred to as the "security" of the database. Usually this is implemented using some sort of access list and passwords, but more complex schemes are possible.

This section provided an overview of the basic issues and concepts of DBMS design. The report which follows will address these basic issues and terms in the microprocessor environment.

III. OPERATING SYSTEM CONCEPTS

Operation of DBMS is often closely related to the operating system. The DBMS must use the operating system or duplicate its functions to perform input/output with various system peripherals such as the terminal, printer and secondary storage (usually flexible or hard disk systems). Of particular importance is accessing data in secondary storage. The majority of the DBMS work involves transferring large amounts of data to and from secondary storage.

The basic function of the operating system is to control the resources of the computer system upon which it is implemented. In general, an operating system must manage the following resources:

1. Memory Allocation
2. Computer or CPU processing time
3. Secondary Storage Allocation (disk or tape media)
4. Peripheral Device Usage (printers and communication equipment)

To manage these resources the operating system usually provides a logical interface between a process (user program in execution) and the rest of the system hardware. In addition it usually handles secondary storage by defining a file management system for the user.

The versions of DBASE II and MDBS III selected for this study work on a particular operating system developed by DIGITAL

RESEARCH called CP/M. The operation of these DBMS rely heavily on many of the basic system functions provided by CP/M and its compatible multi-user (MP/M) and network operating system (CP/NET) counterparts. Both DBASE II and MDBS III will also work on MP/M and CP/NET. Due to some differences in design of CP/M, MP/M and CP/NET both DBMS usually require special modifications to be completely compatible with MP/M and CP/NET. This is due to the fact that MP/M and CP/NET have implemented special functions to handle multiple users which are not currently available on CP/M. In discussing DBASE II and MDBS III it is necessary to frequently refer to the operating system environment under which they operate, thus the following sections are devoted to describing these operating systems.

A. INTRODUCTION TO CP/M

CP/M is described by its developer as a monitor control program for microcomputer system development. The operating system was developed for use specifically on Intel 8080 microprocessor. It will also operate on the Intel 8085 and Zilog Z-80 since these microprocessors are compatible with the Intel 8080 instruction set. The system was designed for one user operating a stand alone microprocessor based system.

In its normal usage, CP/M requires one of the microprocessors mentioned above, 32K to 64K of main memory (32,768 bytes to 65,536 bytes), a terminal, and at least one disk drive (up to 16 are allowed). A typical configuration would be a

Z-80 microprocessor, 64k of main memory, a terminal and two disk drives for secondary storage.

The operating system is loaded from a disk drive automatically during system startup and most of its components reside permanently in memory during normal operation. The operating system is divided into 3 modules: (1) CCP - Console Command Processor, (2) BDOS - Basic Disk Operating System, (3) BIOS - Basic Input/Output System.

The CCP is an interactive interface between the user and the operating system. It provides the basic functions to initiate execution of machine coded programs on the disk and provides basic utility commands such as deleting files and listing files on the disk. This component is not normally used by processes executing under CP/M and the memory on which it resides may be used by these processes for other purposes such as data storage. During normal operation it is loaded in memory just below the BDOS.

The BDOS provides the logical interface between the operating system and the user programs. CP/M provides file management in the form of sequential and random access to files stored on the disks. It also provides logical interface to system peripherals such as a printer and terminal. To accomplish this CP/M provides 39 different operating system function calls. These functions are listed in appendix A.

The BIOS is the interface between the operating system and the physical hardware. CP/M was designed to work on a variety of

hardware configurations. To achieve this portability the BIOS must be adopted to the particular hardware system chosen.

In a typical 64K CP/M system memory will be used as follows. The first page of memory (memory locations 000 - 255) is reserved for CP/M usage. The user program resides in the Transient Program Area (memory locations 256 - 56,320) above this reserved page. The BDOS and BIOS occupy the rest of memory (memory locations 56,320 - 64,535). The CCP occupies approximately 2048 bytes and is loaded below the BDOS. Note that the operating system does not explicitly manage memory. A user program is usually loaded starting at memory location 256. It may use all of memory up to the base of the operating system. Any memory management with this available memory must be done by the user program.

To call the operating system the program uses the jump address at memory location 5 to enter the BDOS. Appropriate parameters are passed in the CPU registers which indicate which function is desired. Additional parameters are passed in other registers depending on what is required for the particular function. Return data such as error codes are returned by the operating system in the the register called the accumulator.

B. INTRODUCTION TO MP/M

MP/M was designed as a multiple programming operating system which was upward compatible with CP/M. This meant that programs which operated on CP/M could also operate on MP/M. In addition to providing all of the CP/M functions, MP/M provided new functions

which allowed multiple programs to time share and communicate with each other on the same microprocessor. These new functions included flags for signaling between processes, queues to pass messages, memory allocation primitives and resource assignment functions.

The minimum hardware environment required for MP/M was an 8080 or Z-80, 32K to 64K of single bank memory, 1 to 16 consoles and 1 to 16 disk drives. In addition there must be hardware to provide an interupt approximately every 1/60 of a second. MP/M can manage up to 8 banks of memory, but when bank memory is used there must a common segment of memory (usually the top 16K) for the operating system.

MP/M organization is similar to CP/M. It resides in high memory and consists of a disk operating system (ODOS and XDOS) and a hardware interface module (XIOS). In order to minimize the amount of common memory certain portions of these modules and system data reside in the first bank of memory below the the common memory. These are referred to as the BNKXDOS etc.

In normal operation each process resides in its own memory partition which was selected by MP/M when the process was activated. At intervals of approximately 17 milliseconds each process would be interupted and CPU time would be given to the next process waiting in the queue. Note that these processes were only temporary and memory is reallocated as soon as they complete their function.

MP/M does have provisions which allow certain processes to remain in memory permanently. These are referred to as "Resident System Processes". When the system is initially started up it can be configured so that certain memory partitions (in common memory) are permanently assigned to these processes. The original use of this was to keep useful utilities in memory rather than loading them from the disk. Later designs used these resident processes to implement a network operating system called CP/NET.

Later versions of MP/M (referred to as MP/M II) provide functions which were important to DBMS. These functions included password protection for file access, concurrency control in the form of file locking mechanisms and the ability to read entire tracks of a disk with one system call. These functions will be discussed more thoroughly in the latter sections of the report. A complete listing of MP/M system functions is given in Appendix B.

C. INTRODUCTION TO CP/NET

Prior to the introduction of integrated circuits, digital processors and associated hardware were extremely expensive even in small computer systems. Systems were usually designed around a single large mainframe computer because it was less expensive to have one large powerful system rather than several smaller systems.

Systems were later expanded for multiple users and multiple processors. This lead to multi-user systems designed around one centralized computer system. With the advent of inexpensive digital processors the situation changed dramatically. Early

systems concentrate computing power in the central processor with only minimal logic in the external hardware and peripherals. Systems can now be cheaply designed with extremely sophisticated logic in the external hardware and peripherals.

The net effect of this is that much of the processing can now be done using logic external to the central processor. Today there are many intelligent peripherals which handle many of the functions formerly relegated to the central processor.

As a result the central processor no longer has to devote as much time to I/O processing, and as a consequence can perform much more efficiently and can do more user application processing. This sharing of system functions could be extended to several processors rather cheaply since individual processors are relatively inexpensive today. This lead to the development of networks and distributed system designs which used multiple processors working together to achieve the power of larger and much more expensive centralized main-frame systems.

Design of these network and distributed systems presented special problems. In a network most of the processing is done by a local processor which calls upon a central processor in the network for basic services such as printing and secondary storage. There must be a logical interface which allows multiple processors to share this central processor. This logical interface must determine which request has priority and keep track of requests which have not been completed. Digital Research

has developed a network operating system which is compatible with CP/M and gave it the designation CP/NET.

CP/NET is designed to work in conjunction with the MP/M and CP/M operating systems. The MP/M system is referred to as the "master" which provides all the public system resources such as secondary storage, printer facilities and (if necessary) terminal input/output. The CP/M systems are attached to the network and are referred to as "slave" processors. CP/NET provides the interface between the CP/M slave processor and the MP/M master processor. With CP/NET, a program operating on CP/M has access to local resources of the CP/M and the public system resources of MP/M.

CP/NET is divided into four modules as follows:

1. NDOS - Network Disk Operating System
2. SNIOS - Slave Network I/O System
3. NETWRKIF - Network Interface Process
4. SLVSP - Slave Support Process

NDOS and SNIOS reside in memory on the CP/M slave processor while NETWRKIF and SLVSP reside in memory on the MP/M master processor as resident system processes. The NDOS provides the logical operating system interface for processes on the CP/M slave processors. The SNIOS provides the interface between the NDOS and network communication hardware. The NETWRKIF provides the interface between the network communication hardware and the SLVSP. The SLVSP provides the logical interface between the NETWRKIF and the MP/M system. Note that virtually any network

communication hardware can be used since only the NETWRKIF and SNIOS have to be modified to handle the specific hardware selected for the system.

The basic sequence of operation for the network is as follows. A program on the slave processor makes a call to the operating system. The NDOS intercepts this call and determines if the call is for a local resource or a public system resource. If it is local then the call is passed to the CP/M operating system. If it is not, then a message is passed to the network via the SNIOS. The SNIOS passes the message on the network in the appropriate form for the hardware. The message is received and passed in appropriate logical form by the NETWRKIF to the SLVSP. The SLVSP then makes a call to the MP/M system based on the request from the slave processor. When the MP/M master has completed the request, an appropriate message is returned by the SLVSP to the requesting slave unit.

The NDOS and SNIOS reside in memory directly below the CP/M operating system. They are normally loaded in memory at system startup along with CP/M operating system or by a special utility loader after the CP/M system has been loaded. The SNIOS contains a table which indicates what system resources are mapped to local resources and which are mapped to public resources. This table can be modified by the user.

The NETWRKIF and SLVSP reside in memory as a "resident system process" of MP/M. Each slave processor has its own

NETWRKIF and SLVSP on the MP/M system. Up to 16 slaves are allowed on the network.

CP/NET operation is completely transparent to the user and the processes which use the CP/M operating system. All system references are made to logical resources which are in turn mapped by the NDOS. The network provides all the same functions available from the CP/M operating system plus it provides an electronic mail system between slave processors. A complete listing of its system function calls is given in Appendix C.

Note that CP/NET provides two basic functions of importance to DBMS implementation. It provides a common logical file system which can be shared by all slave processors. The second function it provides is a division between local and public resources. No other slave processor can access local resources. The local resources can be used independently and are therefore available even when the master is not available due to hardware failure.

Some problems still exist with CP/NET in it current state. The special MP/M II functions such as file locking and password protection are not compatible with CP/M. These functions are not available to the slave urits and thus could not be used a DBMS operating on a slave system and manipulating a database which is a public resource on the MP/M master.

IV. COMPARISON OF DBASE II AND MDBS III

In this section of the report, DBASE II and MDBS III are compared in terms of their basic design and implementation. This section does not address performance of these DBMS's, rather it is a comparison of the data model, features and physical implementation.

A. ARCHITECTURE

The architecture of a DBMS is very important in that it determines much of the final implementation of the database. It defines the interface between the user and the conceptual database, and between the conceptual database and the physical database. These various level of abstraction are costly in that they usually mean more coding and less efficiency. This in turn has a direct impact on the speed and performance of a DBMS.

If a DBMS is designed with a minimal architecture it will, on the other hand, be very difficult to modify. Changing the physical database may entail changing the conceptual database and the user interface. In addition it may be difficult to move this system into different hardware and operating systems.

DBASE II uses a very minimal architecture. There is no host language for this system. The host language and its interface have been combined into one user interface which accesses a conceptual database (relational table). The physical database is simply a collection of CP/M files (each of which contain a

single relational table) mapped directly from the operating system. The file mangagement services supported by CP/M essentially define the physical database. Note that combining CP/M files into a single integrated database is not automatic. The user must manually select which CP/M files will be included in the database which DBASE II manipulates.

MDBS III has a somewhat more complicated architecture. It has a host language and a language interface to the conceptual database. The conceptual database consists of the relationships or sets which have owners and members. This conceptual database is in turn mapped by a separate interface to the physical database. The physical database consists of the data dictionary (which describes the relationships, structure, etc.) and the stored data. The physical database has an interface which maps from the physical database to the CP/M files resident on the disk. This allows multiple CP/M files to be mapped automatically by the DBMS to a single intergrated physical database.

B. DATA MODEL

When properly used virtually any of the three basic data models can be used to represent a database regardless of its structure. Actual implementation of a real world database can be easier with some data models than others. The hierarchical data model works well when the basic structure of the database is a hierarchy or tree. This usually implies traditional relationships

such as 1 to 1 or 1 to many. Note that this scheme would not work well if the basic structure was not hierarchical.

The network data model can be used to model a hierarchical data structure as well as many others. It allows a simple tree structure as well as complex relationships where the relationships are many to 1 or many to many. Such a data model is more general than hierarchical, but is difficult to perform data manipulations on. Queries on the database require navigation thru the many possible relationships in the database. The primary advantage of this data model is that it can represent very complicated data structures with a relatively simple graph representing relationships.

The relational data model is a relatively recent development in DBMS. Most of the successful commercial designs to date have been based on network and hierarchical designs. The relational database is more user friendly (due to the lack of navigation problems and a powerful instruction set) but has one problem. The relationships must be specified by an entry in each relational table which results in duplication of information.

In real world terms this means a relational table would require more storage than an equivalent network model. Later work in relational databases showed that this apparent duplication was the result of not reducing the relations to their simplest forms. The simplest form is a binary relation where there are only two items per relational table entry. The problem is that while databases can be defined it is usually very difficult to reduce

these relationships to their simplest forms, thus some storage is wasted.

The relational database has two very important advantages over other models. Due to its structure it is very easy to develop powerful data manipulation commands. Operations such as a join and sort are relatively easy to implement. The second advantage is that the structure and therefore relationships are relatively easy to change. Modifying relationships in a network data model is normally very difficult, since it will often involve redefining the complex relationships within the network structure.

The data model chosen for a given DBMS has significant implication on the final design. MDBS III is based on a network model and its development stems from the recommendations of the CODASYL Data Base Task Group Report of 1971. The basic building blocks of this DBMS is as follows:

1. DATA ITEM TYPE - These are the smallest unit of the logical structure of the database. Examples of this are individual fields of a personnel record which might include name, age, job etc.

2. RECORD TYPE - These are collections of data item types. For instance, a "city" record type might include name, population, county etc.

3. SET - This is a named relationship between two or more records. Typically there is one owner of the set and one or more members of the set. An example of this would be an owner which is

a country (USA) with set members in the form of states (Ohio, California etc.)

MDBS III has some extensions from the basic network design in that it allows the database designer to directly specify many to many relationships.

DBASE II is based on a relational model which was developed independently by ASHTON-TATE for commercial use (it is not based on any standard). This model is a very simple flat file or relational table. The smallest unit of the system is an individual named field entry in the data table. A record is defined as a collection of these individual fields. A relational table is formed by a collection of records. Note that a relationship is implied between the individual fields of a record. The database is a collection of these relational tables with relationships defined by common fields in each table.

C. EXTERNAL FILE ORGANIZATION

Most practical databases are too large to be stored in the computers memory and when it is not in use it must be stored somewhere. This requires some sort of secondary storage usually in the form of disk storage. In a typical microprocessor system this would be flexible disks and/or a hard disk system. The disk divided into multiple tracks and each track is in turn divided into sectors. These sectors and tracks are of a fixed format and have an absolute physical position on the disk associated with them.

To read data from the disk, the desired track and sector must be found and then the entire sector read. The reason for this is that the error checking scheme used in reading the disk depends on reading the entire sector to perform a checksum. Once the sector is read, the desired data must be extracted by knowing its location or offset relative to the beginning of the sector data. The sectors typically hold from 128 to 1024 bytes of formatted data depending on the particular design chosen.

The file system which uses the disk storage must map its logical records into these physical sectors. This is usually the responsibility of the operating system. In the case of CP/M and MP/M, the operating system provides a logical file space divided into 128 byte logical records. The operating system will provide sequential or random access to these logical records. The file size is limited to 65,536 records or approximately 8 million bytes. Note that individual files are limited to one disk, however, each disk may hold several files if there is sufficient storage space available. The amount of storage available varies depending on disk design.

In order to locate each logical file on a disk, a directory is maintained on the first few tracks of the disk. Normally the first two tracks are reserved for the operating system program. The directory then follows on the next track. This directory identifies the name of each file and what blocks are allocated for each file. These blocks are usually from 1024 to 8192 bytes in size and define groups of logical records (128 bytes each) which are mapped logically by the operating system to the

physical sectors. These blocks are used by the operating system to allocate space on the disk. The size of the block determines the minimum amount of space that must be allocated when a new file is created or extended. This block has no significance to the user since these block allocations are used only by the operating system to keep track of the amount of space left on the disk.

To use the CP/M logical file system, the DBMS must map its own logical records into CP/M logical records. Note that if many different blocks must be accessed to perform a function, the DBMS must make multiple operating system calls (read random or read sequential function calls). To optimize the speed of operation a DBMS must minimize the number of file accesses since the time taken for disk I/O (input/output) is very significant. An additional consideration is the speed with which the operating system can find a given logical record. This will be addressed later in the section on operating system interface problems.

The file organization of DBASE II and MDBS III under CP/M and MP/M are similar in design. DBASE II CP/M files consist of two sections. The first section identifies size of each field in the record, the name of the field and its data type. It also specifies how many records are currently in the file. The second section is the actual data. The data records are placed sequentially in the file in the same order which they were entered (ie. unsorted).

A single relational table is always contained in one CP/M file which limits the size of a DBASE II relational table to whatever one disk will hold. The upper limit for 8 inch drives is approximately 1 million bytes for current designs and an absolute limit of 8 million bytes on a hard disk system due to the operating system random file access limitations. DBASE II will allow different the relational tables to be mapped to different disks which means that the size of the whole database (which is simply a collection of relational tables) is limited to whatever disk storage is available on the whole system.

DBASE II also uses a companion index file for the relational table. This file is created by the DBMS and is similar to balanced binary tree. When searching for a particular record, DBASE II uses a binary search to find a pointer in the index to the first record which meets the criteria. Note that duplicate entries are allowed so each node contains just one pointer to the associated record in the database. Searching this index is usually quicker than searching the entire database because it is organized for binary search and is much smaller than its associated database. Any number of index files may be created for a given relational table. Either single or multiple fields may be used to form each index file. Index files which are designated as "in use" are automatically updated as data is added, deleted or modified. Note that there is a limit of seven "in use" indices which means only seven indices can be automatically updated.

If DBASE II is using the index to search a database, it must first move the index into memory from the CP/M file and then

search the index for the desired record. The index identifies which record in the database is desired. Then DBASE II computes the location of this record within the CP/M file and gets the CP/M record or records (using random access) which contains the correct DBASE II record.

MDBS III has only one type of file associated with the database. The database however can be automatically mapped to several different CP/M files which eliminates operating system and disk system size limitations. The key part of the file is referred to as the main area of the database. This part of the file contains the "data dictionary" which identifies the structure of the database and contains MDBS III system information such as access list and passwords. The file is divided into individual pages (usually 512 bytes) which are moved in and out of memory by MDBS III. This paging is the method that MDBS III uses to manage memory.

The data dictionary and DBMS information is located in the first few pages of the main file or area. The remaining pages are used for data storage. Each page is filled with individual records till there is not enough room for another complete record. Records are not placed over page boundaries because individual pages are moved in and out of memory during DBMS operation. To move one page into memory requires four system calls (4 X 128 = 512).

To find a particular record, MDBS III would normally navigate through the database finding pointers from owners to

members. These pointers are mapped by the DBMS to the physical database page (512 bytes) which is in turn mapped to 4 CP/M records within a CP/M file. To access the data in the CP/M files, MDBS III would normally use the random access function of CP/M to minimize the time taken to access the desired data.

D. OPERATING SYSTEM INTERFACE PROBLEMS

Today virtually all computers have an operating system of some sort. This process provides a logical interface to the hardware and controls the general sequence operations in the computer system. It simplifies the program interface to the hardware and provides some portability since the operating system logical interface could be extended to several different systems. But just as it eliminates some problems, it also creates some new problems. The operating system can add a lot of overhead CPU time and thus slow down the operation of a DBMS. In addition, it may not provide efficient access to the system resources.

In some ways it would be more efficient to let the DBMS directly interface the hardware. Secondary storage is usually disk media of some sort. Efficiency of this system can vary widely depending on the algorithm used to map and access the physical records contained in it. If this decision were left up to the operating system it could be very inefficient and thus slow down an otherwise good DBMS design. A DBMS must be able to access system resources efficiently. This is due to its basic purpose which is to access and manipulate large amounts of data.

A second and equally serious problem caused by the operating system is related to its basic task of scheduling resources and working processes. Many multi-tasking and multi-usersystems use time slicing to schedule processor use. Certain operations of the DBMS should not be interupted arbitrarily just because its time period has been used. A typical example of this is concurrency and synchronization control within a DBMS. Locking schemes used to insure concurrency must take into consideration possible interuptions of the locking algorithm.

In both of these cases the DBMS must bypass the operating system or devise a method of achieving its functions within the operating system. One alternative to this is designing the operating system with specific requirements of a DBMS in mind. This could be achieved by designing into the operating system certain primitive operations that the DBMS needs. Some examples of this would be operating system functions which would lock individual records or whole file. MP/M II has some of these functions which will be addressed later in the section on concurrency and synchronization.

First, let us address usage of secondary storage via the CP/M operating system. This system divides a file into logical records 128 bytes long. When reading files in non-sequential manner (referred to as "read random") the record desired is identified by a 16 bit integer. This limits file sizes to 8 megabytes. In order to implement larger databases it may be

necessary to define several different files to get enough storage for the entire database.

Storage on the disk is physically divided into sectors on each track. These sectors typically contain 128, 256 and 1024 bytes in current 8 inch disk drive designs. This means that the operating system system must read the entire sector to get one 128 byte record. In addition it must map the logical records into the physical sectors. Typically this means the system will buffer the entire sector and perform disk reads only if the logical record is not currently in the buffer.

This implies that data which is written to a track is also buffered. The new data is not written to the disk till the buffer overflows. This condition occurs when a new sector is read or written. In addition, the directory entry on the disk is also stored in memory. This directory entry indicates where on the disk the file is located. Note that if a system failure occurs before the buffer is flushed (written to the disk), data is lost. In practical terms, a DBMS operating on the CP/M has no indication when data is actually written to the disk unless it bypasses the normal operating system function calls.

On the early versions of the MP/M operating system, this problem also occurs. However MP/M II has a system function call which will explicitly flush the disk buffers (called "flush buffers"). If this MP/M II function is not used or is unavailable, the DBMS must ensure that a database file has been updated. To do this the DBMS must mark the file somehow to

indicate that it was opened and updated. Upon closing the file it must again mark the file as closed when it has performed its last operation. This is the basic method used by MDBS III to determine if a file was properly opened and closed. DBASE II has no corresponding function for this and thus may loose its consistency (updates are only partially completed) without the user being aware of it.

Another problem is the algorithm used by CP/M to perform the logical to physical mapping of records on the disk. In a normal eight inch single density floppy disk system the sectors are mapped with a skew factor of six. What this means in practical operation is that even though the records are mapped one to one to the sectors (each sector is 128 bytes) the sectors are not sequential on the disk. To improve system performance the next logical sector is six sectors from the last one. This gives the computer time to absorb the previous data and be ready for the next logical sector as it rotates under the read head. If the system misses this sector it must wait till it comes around again. This skew factor is optimized for general operation and will not not necessarily be optimal for a DBMS.

There is also an additional complication in that systems like CP/M do not move files around as others are deleted. It merely looks for the first available space on the disk when it is adding new files. The allocation blocks are usually anywhere from 1024 to 8192 thousand bytes in size. This means that only these portions of a file will be together. The rest may be located any

where on the disk. It would be useful if the DBMS could control the disk block allocation and sector mapping so that it could be optimized for DBMS operation, but at present there is no way this could be done except by bypassing the operating system. These complications of access within general purpose operating system make disk access one of the most time consuming parts of DBMS operation.

In MP/M II and certain operating systems which emulate it, the entire disk track can be read at once which eliminates waiting for a particular sector. This can improve performance because rather than doing several individual sector read operations (with its associated head positioning time), the system can read the sectors sequentially as they rotate under the read head. Note that currently, neither DBASE II or MDBS III take advantage of any of the extra functions offered by MP/M II. This will probably change in future releases.

E. DATA DEFINITION CAPABILITIES

Since the DBMS is used to define a logical data structure it must have some language and terminology which can be used to specify the database. Also since the actual database will exist on physical storage media there will be some options for the user to determine how the data will be stored on the chosen physical media.

The DATA DEFINITION LANGUAGE (DDL) must be able to specify the data structure of the entire database as well as individual records. Usually this means that it will handle a record type

structure, character string structure and in some cases other structure such as an array. In addition it must be able to handle various data types such characters, integers, real numbers and boolean. Finally it must have a way of expressing relationships between the data elements which in itself is a structure.

DBASE II uses a single unified language developed by ASHTON-TATE for use with the system. Certain portions of it are used to create the database and its structures. The relational database structure is relatively simple and as a consequence the language is relatively simple. The command for creating a database is "create" which then prompts the user for the field names, their size and data type. The data types are limited to character strings, numeric and boolean. This creates the traditonal relational table. Note that a separate CP/M file must be created for each relational table and that the CP/M file is limited to one CP/M disk. Index files may also be created for each relational table. These indexes may be created with the relational table or may be added later. The choice is up to the user.

MDBS III has a separate DDL which is much more complex due to the complexity of the network structure. In addition, there are some extra instructions to implement access lists, file encryption and control of certain physical aspects of the database. The basic operation of the DDL is simply to invoke the DDL program which analyzes the user DDL specification. If there are no errors, the database is created and initialized.

The DDL specification is organized much like a high level language program and constitutes a complete description for the database. It consists of six basic blocks as follows:

1. Identification section
2. User section
3. Area section
4. Record section
5. Set section
6. End section

The identification section names the database, sets the page size and size for the database, and sets system defaults. The user section contains the user names and their passwords. The area name identifies the different CP/M files which are mapped from different disks to the integrated database. This section is optional since the database does not have to map to more than one CP/M file. The record section identifies the structure of the the individual records and types of data within it. This section is used also to specify if data is encrypted and to specify range bounds or consistency constraints on the data. The set section is used to specify the owners and members of each set. It also identifies the type of relationships between owners and members (1 to 1, many to 1, 1 to many, etc.). The end section merely indicates the end of the database specification.

Extra restrictions for access are allowed at virtually any level within the DDL specification, provided it is a subset of the one specified in the user section. The creator of the

database can specify separate access lists for database areas, sets, records and data items.

The identification section of the MDBS III DDL sets two important database parameters. It allows the user to specify the size of each page in the database. The default size is 512 bytes with a minimum allowable size of 256 bytes. The user can change this to optimize record storage. Records cannot cross page boundaries, thus the user should specify page sizes which hold an integral number of records. If this is not done then some space will be wasted. For example if one record occupied 272 bytes and each page was 512 bytes then approximately 240 bytes will not be used on each page. The second parameter is the size of the database in terms of the number of pages. The page size and number of pages determine how many records can be stored.

Certain features of the DDL specifications related to database structure should be mentioned at this point. The sets, members and owners need not be absolutely fixed when the database is created. Insertion of members and owners can be specified as manual vice auto. This means that they can be inserted or deleted by the connect/disconnect commands listed in Appendix D. Set retention also has two options referred to as "fixed" and "optional". If retention is optional, the relation or set can be removed. These options in effect allow the database structure to be modified dynamically. Note that these options must be specified when the database is created. They cannot be added

later. Note that DBASE II does not have this limitation since new relations can be added at will.

There are some other features of interest in the MDBS III DDL. Members and owners of a set can be in the following order:

1. first in - first out
2. last in - first out
3. next
4. prior
5. sorted
6. unsorted

This order will be maintained automatically as new records are inserted into the database. In addition, a calc key can be specified for each record. When a calc key is used, a record can be located directly from an index rather than searching the database.

F. DATA MANIPULATION CAPABILITIES

The Data Manipulation Language (DML) is used to manipulate, insert and delete data into the database. It basically consists of two parts. The first part deals with the functions necessary to answer questions or queries. The second part deals with the various routine functions not associated with answering questions. These functions include creating and inserting new records, deleting records, finding particular records, modifying records and utility functions. MDBS III has a well defined DML, but DBASE II has integrated all these functions into a single language. In this section, the discussion will limited to the

routine or non-query functions of DBASE II and MDBS III. The query or question functions of these DBMS will be discussed in the section on interactive query capabilties.

The DBASE II DML commands are loosely divided into three groups:

1. search commands - find a particular record based on a criteria or its position in the relational table

2. modify commands - delete or change records

3. insertion commands - append or insert new records

These commands are summarized in Appendix E which includes all commands, operators and functions available in DBASE II. Unlike MDBS III, DBASE II has no explicit navigation commands to work its way through the database. Instead, the relational table of interest is simply selected by the user.

Typical search commands are find and locate. Find uses the index to position the record pointer to a particular record which has a field or fields of some selected value. A typical use of this would be to find a particular name in personnel records. The locate command does not use the index for the search. It merely starts searching the relational table sequentially until it finds the desired record.

The modify commands have several interesting capabilities. The browse command allows the user to look at a relational table and modify entries as desired. This command is full screen oriented and thus works like a window on the database which can be placed anywhere within it. The Update command allows the user

to update one relational table from another relational table. The replace command allows the user to change selected fields of a relational table based on any arbitrary criteria. The delete command allows the user to delete records from a relational table based on some criteria.

The insertion commands consist of the append and insert command. The append command allows the user to append records to a relational table from another relational table or text files in the appropriate format. The text file format is compatible with many high level languages such Cobol, PL/I, Fortran, etc. The append command also has an interactive mode which will allow the user to add records in a full screen editing mode. The insert command allows the new record to be inserted into a particular position (selected by user) within the database.

DBASE II has no separate host language. The capabilities associated with the host language have been integrated into the DBASE II language which is interpreted interactively. It provides such features as interactive user input/output, console display, printer interface, file input/output, arithmetic operations and modular programming constructs (if-then-else, do-while, case or switch). This langauge does have some problems common to some other high level languages. The "memory variables" are all global. Subroutine calls are allowed but they provide no parameter passing other than through global variables. Record structures are not allowed other than those directly tied to the

database. The only data types are boolean, numeric and character string.

This integrated language does provide some special benefits in the form of special functions which make the process of modifying and adding records to the database much easier. DBASE II has many single line commands which perform functions such as screen oriented terminal input/output, updating one relational table from another, appending to one relational table from another etc. These one line commands perform functions which normally take a complete program. To make this even easier DBASE II provides a command which will check the syntax of commands interactively. With this capability, the user may check his program (prior to execution) for syntax errors.

DBASE II uses a run time interpreter which simply decodes and directly executes command lines as they are entered. No linking or compiling is required. In addition to typing in commands directly the user may create command files with a text editor. These command files can be executed just as if they were typed in by the user. These can be used to form programs for commonly used applications. In addition subroutines can be defined since command files can call other command files.

The data manipulation language for MDBS III is referred to as DMS by its creator. These commands follow very closely the commands developed in the CODASYL Data Base Task Group Report of 1971. The DML consists of fourteen command groups of which six

groups are basically utility and system commands. The remaining eight are the basic manipulation commands which are as follows:

1. Find commands
2. Retrieval commands
3. Modify commands
4. Assignment commands
5. Creation commands
6. Connect commands
7. Disconnect commands
8. Deletion commands

These DMS commands (Appendix D) are available as separate modules which are linked into a host language. Many different host language interfaces are available such as C, Fortran, Cobol, Pascal and Basic. For testing purposes PL/I-80 from Digital Research was chosen. This language is a subset of PL/I as defined by the ANS PL/I Standardization Committee.

To use the DMS a program is written in PL/I which refers to these DMS commands as an external functions. After the program is compiled, it is linked. This linking process ties in the externally referenced functions and procedures from the selected library files. The linker then creates the final machine code program which can be executed on CP/M. The host language provides input and output to the console and files which are not part of the database.

The MDBS III DMS commands are limited to a few basic functions (query functions are provided in a separate interactive

QRS). These functions are essentially ones that follow the network graph (navigate), modify a record, delete a record, insert a record and locate a particular record based on order or a match on a particular field. This is a limitation inherent in any network DBMS design.

G. INTERACTIVE QUERY CAPABILITIES

The function of the QRS or query language (which is part of the DML) is to provide the user with a means of getting information from the database. This could include a variety of possible questions. Examples are printing out a list of database entries, locating and listing selected records, generating reports, describing the structure of the database and summarizing data. Generally, the DBMS which are based on a relational model provide a much more powerful query language. This is due to the fact that the relational table structure is very simple and thus much easier to manipulate than the complex graph structure used in network DBMS.

The QRS or query language for a relational model can be broken down into two groups: (1) algebraic and (2) predicate calculus. In an algebraic langauge we specify some operations on a relation and take the result as our answer. Typical operators are union, select, intersection and join. Note that join and intersection are actually a combination of simpler operators. In predicate calculus based query languages we express what conditionsthe resulting answer should meet.

DBASE II is apparently developed enough to perform all the functions considered necessary for a "complete" relational DBMS. The term "complete" means that it has at least the minimum number of operators and functions necessary to perform all possible relational manipulations. DBASE II is not exclusively algebraic or a calculus based language. There are some explicit algebraic operators such as join and sort. The display command however resembles a calculus langauge expression in that the user defines what records to display on the terminal by stating a criteria they must meet.

The basic DBASE II query commands of interest are sort, join, list, display and report. Sort will take a relational table and sort it on a particular field. This sorted relational table is placed on the disk as a new CP/M file. The join command will take two relational tables and combine them to create a third relational table based on a match in one or more fields. The display and list commands are similar commands which will describe the structure of a selected relational table and display records in the table based on one or more criteria. The report command is essentially a report generator which allows the user to select fields from a relational table for a report. The individual fields of the report can be formatted as desired with special headings, selected margins and page length. In addition numeric fields can be divided into subgroups and totaled. The format selected for the report is stored on disk and can be recalled at will.

The query language of DBASE II is much more powerful than the MDBS III QRS. The reason is the relational structure used by DBASE II makes it easy to implement commands such as join and sort. Note that sort command allows the database to be sorted after it was created. MDBS III allows records to be sorted and indexed (calc key) but this must be specified when the database is created. This is one of the unique advantages of the relational model which allows extensive modification or reorganization of a currently existing database with very little programming.

The MDBS III QRS has three expressions available for queries. The first one is the list command. It is similar to the DBASE II list command except that there is an extra expression required to navigate through the network database. The other two commands are similar to the list command and are called write and spew. These write data to the disk rather than display it on the terminal. Write and spew allow specially formatted reports and data to be written to a disk file. They are not, however, as flexible as the DBASE II report command.

An additional command of significance is available in the MDBS III QRS which is called display. This command is similar to the DBASE II display command in that it allows the user to determine the structure of the database. It will identify sets, members and owners as well as the individual structures of each record.

Both MDBS III and DBASE II offer what is referred to as "extensions" to the basic query language. These extensions are such things as arithmatic computation, printing out selected data, utility commands (open/close files, set system parameters, etc.) and aggregate functions (average, sum, max, min etc.). MDBS III has more aggregate functions than DBASE II in that it will perform statistical analysis on the selected records. This function is not available as a single command with DBASE II. A complete listing of query commands for both DBMS is contained in Appendix D and E.

Both DBASE II and MDBS III QRS functions are implemented interactively. This means that queries are interpreted and executed as essentially one operation rather than going through a sequence of compilation, linking and execution. These functions can be combined into "macro" instructions. This allows the user to replace a group of intructions with a single command. This is convenient if particular instruction combinations are used frequently either by direct input from the terminal or within a query program.

H. CONCURRENCY AND SYNCHRONIZATION

As mentioned previously, there are special problems which occur when a database is accessed by multiple users. When using DBASE II and MDBS III under MP/M and CP/NET the problem of concurrency and synchronization occurs.

The present versions of CP/NET and CP/M and early versions of MP/M had no provisions for concurrency control. To allow

concurrency control under these circumstances, MDBS III devised a proprietary file locking mechanism which allows the user to actively or passively lock individual MDBS III records. If a record is passively locked it can be read by any user. If it is actively locked it cannot be read by any other users. A record which is actively or passively locked cannot be modified by other users. When the DBMS is operating, the record which is currently being used (current of run unit) is always passively locked. The user may actively lock the MDBS III current of run unit, record type and/or a set.

In MDBS III if a file is opened and only passive locks are used then any number of users can share (ie. read) the same records simultaneously. In this condition no user can modify a record till all of the other passive locks have been removed. When this condition occurs or active locking has been used the system provides an instruction (MCC - multiuser contention count) which allows the user to control how long the DBMS will wait when locked out before reporting a failure to access the record. This locking enforces concurrency and synchronization since other users cannot access and modify the record until the first user is done with it.

Note that the MCC instruction provides a means of preventing "deadlock". The basic deadlock situation occurs when two users attempt to simultaneously lock and access the same two records which we will refer to as "A" and "B". One user locks "A" and the other user locks "B". Then they both attempt to lock the second

record which is impossible since it is already locked. These user processes would then wait for the record to be unlocked. This will never happen since one is waiting for the other. With the timeout set by MCC, an error will be generated which will can be used to terminate the record locking and thus allow at least one of the users to continue.

Although MDBS III provides its own method of concurrency control, there is an operating system alternative. The MP/M II operating system does provide concurrency control in the form of system primitives which allow file locking and verify before write algorithm. Note that these mechanisms are completely supported by the operating system and are essentially transparent application programs except that they must select which algorithm they will use.

The first method of concurrency control allowed by MP/M II allows locking of files. This method locks out all other users until the process is finished with the file. The second method allows a file to be opened in read/only mode which guarantees concurrency since the file cannot be changed. The third method allows shared read/write access by two different modes. In one mode the user can lock and unlock individual CP/M records in the file thus ensuring concurrency. In the second mode the record is not locked. Instead the user provides the system with the old version of the record and a new version. The operating system compares the old version with the one currently on the disk. If

it is identical, the new record is written. The operating system does not allow interupts during this verification.

This shared read/write concurrency control of MP/M II creates a problem in that it places a large overhead on the operating system. It must maintain a list of all locked records and update the directory information to all processes which are using the file. This means that operations such as adding records and reading records requires the system to check all open user files to see if it affects any other processes which are using the file. If speed of operation is important, files should always be opened in a locked mode or read/only mode.

Currently, DBASE II and MDBS III do not use the locking mechanisms of MP/M II. MDBS III still has its own locking mechanism which is independent of the operating system. DBASE II currently does not use any mechanism to enforce concurrency.

I. TRANSPORTABILITY

Ideally a DBMS should be very portable so that it could be used on a variety of computer systems. This would eliminate dependence on a particular hardware system and operating system. It would also be easier to integrate it into existing systems and adopt new systems. No DBMS can meet all of these needs. The issue can be addressed on three levels:

1. Portability of applications programs in high level language.
2. Portability of DBMS in high level language.
3. Portability of DBMS machine code.

At the lowest logical level is the machine code of the DBMS which is given to the end user. One would expect that this code would not be very portable because it depends on a particular environment. The popularity of the CP/M operating system, however, has made DBASE II and MDBS III useful on a variety of computer systems. The only problem in this area is the use of full screen editing capabilities associated with the terminals used. This problem has been largely eliminated by development of programs which customize the terminal driver for the particular terminal used. The other alternative is to use line oriented input and output which will work on virtually any terminal.

At the next logical level is the coding used to develop the final machine code. This could be assembly language code or some higher level language. Here the two criteria of portability and speed conflict with each other. The use of a high level language makes the code very portable since all that is needed to move it to another system is an appropriate compiler. The language chosen should be reasonably efficient and have suitable structure for development of the DBMS. One such language is C which is available on most of the commercially available computer systems today.

If a high level language is chosen and used, another problem occurs. The high level language usually means some sacrifice in speed because it is difficult to make code developed this way as efficient as assembly language. Typically the approach taken

would be to develop the initial system using a high level language and then optimize the final machine code.

Initially DBASE II and MDBS III were apparently written in assembly language which was not very portable but allowed the best efficiency and speed of operation. It is also possible to translate 8080 and Z-80 assembly language to other systems such as the 8086 which uses CPM-86, the 16 bit version of CP/M. There is some indication that new versions of these systems may be developed in high level language to shorten implementation time.

At the highest level is the user programs written for a particular DBMS. If the DBMS is available on many different systems, then applications programs could be used on many different systems. This has often not been possible in the past since most of the DBMS were developed for a particular system. In many cases they could not be used on different computers even if the computers were made by the same company.

MDBS III is currently available on most of the popular micro and mini based systems. This is due to the fact that it has been adapted to a number of computer systems by its developer. The DBMS machine code for these various systems vary widely but at the user level the DBMS function calls and host languages interface are virtually identical. DBASE II is currently being offered on CP/M and compatible systems. Since it is interpreted at run time, only the interpreter is modified to run on other systems. The langauge of DBASE II is not changed.

J. INTEGRITY AND CONSISTENCY

Regardless of the nature of the data in a database it is useful only if the data is reliable. The DBMS must have some ability to determine if the data entered is valid and appropriate for the data item under manipulation. If it is not then the system must reject it. Usually this involves checking the input data type and its range. The range can be checked against fixed upper and lower limits or it might be checked against some other value in the database.

DBASE II has rather limited input consistency checks. Basically the user can only specify the type of data input which is checked when data is inserted. If more restrictive range checking is needed it must be specifically written into a program which checks the data values prior to inserting them into the database. This could be done just by reading the data under program control and checking the value to see if it is in the desired range. DBASE II does not offer any automatic range checking.

MDBS III checks both data type and range limits on all insertions. This range checking is automatic and is programmed in by the DDL when the database is created. It is also possible to program in other types of range checking using the host language. This involves writing in PL/I a routine which checks the range of the data.

K. SECURITY

One very important feature of a DBMS is the ability to provide security. This security to be useful and effective must consider several things:

1. Prevent unauthorized access to data in secondary storage.
2. Prevent unauthorized modification of data in secondary storage.
3. Prevent unauthorized access to data when the DBMS is in use.
4. Prevent unauthorized modification of data when DBMS is in use.
5. Security should be selective in that it will control access down to the individual fields of a record.

Absolute security requires that the database be physically protected and have adequate back-ups. In addition the operating system must protect the database records from unauthorized access by processes operating on the system. This protection would require some sort of password or key used to authorize access. Even when this is done the DBMS must provide appropriate security of its own when it is in operation to ensure complete protection.

Operating systems such MP/M II provide password protection to the files. This password can be used to protect the file from being read, written and/or deleted. This would be useful to protect the file from being read, modified or erased by a program or user not within the DBMS. CP/M does not provide this feature, but it is possible to control physical access to the computer

system. CP/M systems are usually small single computers with only a terminal for input. Physical access to this system could easily be controlled by a locked door.

What about the DBMS itself ? Ideally it would have some sort of access key which can control who can read and separately who can write records. In addition, it is desireable that this control could be explicitly used or not used down to the level of individual fields within a record.

DBASE II does not provide any explicit security. Files from this system do not require passwords or keys to use them within the database. If the files are examined by using system utility programs it is possible to extract most of the information in the file. It is possible to put password protection within the command files, but these could be easily circumvented by reading the command files (DBASE II programs).

MDBS III provides a rather elaborate security scheme. If specified by the DDL, part or all of the database can be encrypted. Its use is transparent to the user within MDBS III since it will automatically code or decode as required. In practical terms it would probably provide adequate security for most users, since the encoded data would be hard to extract from the disk using without MDBS III.

In addition to file encryption, MDBS III allows read and write access lists (name and passwords) for the whole database or parts of it. The DDL allows the access clause to be placed at virtually any level of the database including individual items

within a record. This is convenient since it may desireable for a user to have access to all but one entry in record. An example might be a list of employees were it is desired to keep the salaries of individuals private but allow access to name, address etc. Separate access list can be specified for the database, an area within the database, a set, a record and data item. The basic mechanism for access control is that each user must enter an appropriate name and associated password for any information that is accessed. If the name and/or password are not correct, no information will be provided by the DBMS. This password protection works with all software associated with MDBS III DBMS.

L. CRASH PROTECTION AND RECOVERY

One important aspect of a DBMS design is its ability to protect itself from damage in the event of hardware failure and to recover from any damage which occurs to its storage media. Typical problems which occur are failure of the disk media such as simple mechanical damage to the disk, power failures which temporarily stop the computer system, transients in the power supply which cause data in memory to be scrambled.

Most DBMS designs use a transaction logging scheme coupled with a back-up copy of the database to reconstruct a damaged database. In the event of a power failure it is possible to reconstruct the transactions and bring the back-up copy up to the same status as the original database before the failure. Note that reconstructing the database from the copy requires generating another copy which could take quite a bit of time. The

best designs should use back-up and reconstruction only as a last resort.

DBASE II has no special functions for crash protection and recovery other than its built in instructions which allow all operations and transactions to be echoed to the printer or a disk file. If a back-up copy of the database was available it is possible to go through the transaction by hand and reconstruct the database.

MDBS III has several special features which protect it from a system crash and allow it to recover from a damaged database. Note that these functions are made available through a special option (which can be purchased separately) called RTL/RCV. There are basically two methods, one prevents damage to the database and the other reconstructs a database once it has been damaged.

The first method is referred to as "pre-image posting". The user declares a pre-image file which stores all updates and then causes an update to the database from the pre-image file. The sequence starts with a pre-image file declaration (PIFD) followed by a command to begin the transaction sequence (TRBGN). When the command is given to complete the transaction sequence (TRCOM), the pre-image file is posted to the database. It is also possible to abort the update by using the transaction abort command (TRABT). Note that this abort can be done only for transactions since the last TRCOM command. If a failure (power failure or hardware failure) occurs the database is undamaged and all that

is lost is the transactions which occurred since the last transaction commit (TRCOM).

Should a failure occur during an actual database update (during disk write) or as a result of media failure, then the *second method must be used*. With this method a transaction file is maintained by the DBMS which is used by another program called RCV. This interactive utility program will take a back-up copy of the database and use the log transaction file to make an updated and corrected database. At the users option some of the logged transactions can be eliminated. This would be useful if erroneous transactions were performed. Note that the entire database must be duplicated before using RCV so that there is at least one unmodified copy available if another system crash occurs. The transaction log file is automatically maintained by the MDBS III but maintaining a back up copy of the database is the responsibility of the user.

M. SUMMARY OF GENERAL FEATURES

There are many general features of a database not directly related to design. These include ease of use, available extension packages, limits, etc. This section provides a comparison of these general features.

DBASE II currently has a list price of $700. and has one extension called zipscreen which is sold with the package. Zipscreen is a utility which creates DBASE II command files based on user input. The purpose of this is to simplify production of screen oriented input and output to the database.

MDBS III offers the DDL, DMS, RTL/RCV and QRS as a package. Extensions to this basic package are the DMU and IDML. DMU allows the user to modify certain parameters in a database which has already been created. Its basic purpose is to allow the database size to be expanded and to change the access list (names and passwords). IDML is an interactive version of the MDBS III DMS which can be used to trouble shoot application programs. The cost of the whole system is approximately $4500.

Another area of comparison is range allowed on the basic data types used by the DBMS. This comparison is based on the worst case limits for MDBS III with its host language interface and operating system. For DBASE II the limits are determined by the interpreter it uses and the CP/M operating system. The limitations and constraints on each system are given in Appendix F. It should be noted that many of the limits for MDBS III are imposed by its host language.

Appendix G contains a comparison of DBASE II and MDBS III in tabular form. The purpose of this table is to summarize many of the features and design characteristics of both DBMS.

V. COMPARISON OF PERFORMANCE AND USER INTERFACE

In this section, a direct comparison is made between DBASE II and MDBS III in performing identical problems. To perform the comparison three databases were selected. Then several basic problems were chosen for each database which were done using both MDBS III and DBASE II to allow direct comparison of peformance and programming requirements. This section of the report describes how the test was performed and its results. In addition some comments are made on user interface and ease of use.

A. BASIS OF EVALUATION

The three databases selected were derived from databases already in use at the Naval Postgraduate School. In the following paragraphs the fields and contents of the database are described along with the data manipulation problems selected to compare performance.

The first database is an inventory of equipment used in the Physics department. This database contains approximately 1500 records and the fields are as follows:

field number	name	width
1	item number	5
2	item name	16
3	model number	16
4	serial number	16
5	manufacture code	5
6	manufacturer	9
7	year made	3

8	year purchased	3
9	cost	4
10	fund code	2
11	plant account	13
12	stub number	10
13	person responsible	6
14	location	6

The second database is a listing of students in the engineering curriculums at NPS. This database contains approximately 600 records and the fields are as follows:

1	name	24
2	rank	4
3	service	4
4	country	4
5	designator	4
6	section	4
7	graduation date	4
8	section leader (y/n)	1
9	option	5
10	curriculum	3
11	degree	1
12	student mail code	4
13	phone number	7
14	address	45
15	report date	6

The third database is a listing of library periodicals and references. This database contains approximately 700 records and the fields are as follows:

1	reference number	6
2	cross reference (x)	1
3	type of publication	1
4	type of binding	3
5	binding date	4
6	country published	2
7	language used	4
8	subject code	16
9	account number	4
10	microfilm (M)	1
11	translation (T)	1
12	depository item (D)	1
13	frequency of publication	2
14	retention time	1
15	display codes	2
16	current subscription (+)	1

The basic problem presented was to create a database in MDBS III and DBASE II and then perform four data manipulation problems on each database. The problems were selected on a functional basis to simulate data manipulation problems which commonly occur. A detailed description of them is included in Appendix H. The following is a summary of the data manipulation problems:

1. Select the records which meet a specific criteria and display them on the console. Create a printed report from the selected records. This is perhaps the most common data manipulation problem. It involves locating particular records and preparing a report.

2. Create a second database and perform a join operation using the original database. An example of this might be preparing a parts order for a customer. The customer has a list of part numbers which is joined to a companies database of parts to create a list which lists part number, price, location and availability of part from inventory.

3. Create an output file which is a projection (subset) of the original database. This problem might occur when some fields are no longer used and the user wishes to compact the database.

4. Create a new database from the original database. The new database is sorted on one field. This problem often occurs as data is appended to a database in random order (or in some order which later is not used) and the user would like to organize it on a particular field.

These problems are of course arbitrarily and do not completely test all aspects of each database. They were picked simply to provide a direct performance comparison of speed and usage of the DBMS.

These problems were given to students at the Naval Post Graduate School taking a course on Data Base Management Systems. Most of the students were working on their Masters Degree in Computer Science. The tests were performed on an Altos multi-user system with a hard disk and Altos single user system with a flexible disk. As part of the test, the students were required to develop programs to perform each problem and then time how long each DBMS took. After completing the problems, the students answered a questionnaire designed to access what they liked and disliked about each DBMS.

Since DBASE II does not have facilities for concurrency control and crash recovery these facilities were not used in MDBS III when performing the tests. This would allow both DBMS to operate at their maximum speeds. In addition, the students attempted to optimize their programs as much as possible. The best combinations of commands were used in determining the average execution time for each problem.

B. RESULTS OF PERFORMANCE EVALUATION

The problems for each database were performed using both DBASE II and MDBS III on a Altos multi-user system (MP/M operating system) and a Altos single user system (CP/M operating system). The average execution rate for each problem using DBASE

II and MDBS III is summarized in Appendix I (The avaerge execution rate is the number of database records the DBMS processed in one minute). It should be noted that DBASE II was significantly faster than MDBS III in all cases since it took approximately 60 percent of the time that MDBS III did to perform the same problem. These results should be regarded only as an indication of performance since they are dependent on the nature of each problem and the approach taken to solve it.

Some additional facts were noted by the students during the testing. MDBS III took much longer than DBASE II (approximately 3 times as long) to insert unsorted data into an empty database. In cases were MDBS III sorted the data as it was inserted into the database, the time for DBASE II to append and sort the data (recall DBASE II always inserts data without sorting) was roughly comparable.

During normal operation certain sorting options degraded the performance of both DBMS. Using an index with a DBASE II relational table will significantly increase the time of most data manipulation problems. Note that the index is not needed except when the "find" command is used. Its use other than when absolutely necessary will seriously affect performance. Addition of more indexes (up to seven are allowed) will cause additional degradation.

MDBS III performance is degraded when set members are sorted. There is an apparent degradation of speed in both the insertion and extraction of data when one or more sets are sorted.

The degradation in both cases is probably due to the overhead required in ordering the records. When an index is used withDBASE II the relational table is not sorted, instead, the entries in the index table are ordered. Each time a record is retrieved, the index must be searched to locate the next desired record. It is possible that the overhead of these sorting methods actually degrade overall performance in some problems.

The user interfaces for the two DBMS's are distinctly different. Most users felt that DBASE II was the easier to use than MDBS III regardless of the type of data manipulation problem. The reasons are summarized as follows:

1. DBASE II uses one integrated language for all applications which is easier to learn.
2. The relational structures are easier to understand than the network.
3. DBASE II is interactive and full screen oriented
4. The command set is much more powerful than that available with MDBS III.
5. The run time interpreter eliminates compiling, linking etc.

The users felt that MDBS III was more sophisticated and capable of more complex tasks than DBASE II but that it was very difficult to take advantage of its capabilities. The reasons for this are summarized as follows:

1. MDBS III has three separate languages (DDL,DML and QRS) plus the host language which the user must know.

2. Applications take to long to develop due to the sequence of compiling, linking and execution.

Most users felt that MDBS III was intended to be used in professional applications where reliability and security were important and there are full-time programmers available for program development.

VI. CONCLUSIONS

No absolute conclusions can be drawn from this comparsion of features and performance of these two DBMS, as this was not the purpose of the study. The basic goal was to contrast the differences in design, features, user interface and performance.

It is apparent that DBASE II performs very well for small systems. The user interface is easy to use because it is interactive and full screen display oriented. In addition, it uses one common integrated language with a powerful instruction set oriented to data base management. Many of the normally required manipulation functions have been integrated into single packaged commands which have been optimized for maximum speed. Finally, it is easy to understand due to its relational structure. Users can orient themselves and use the system in a matter of hours.

DBASE II does have some disadvantage in that it lacks the sophistication to work well in large scale applications. When a database is large and complex with many users there must be protective functions for integrity and security. In addition there must be some form of concurrency control if many users are to simultaneously access and share the same data.

MDBS III would work best in these large scale applications since it can control access, recover from system failures, provide automatic integrity constraints and ensure concurrency.

In addition its network structure can handle complex structures common to large organizations. Such applications would be sufficiently important to merit the extra cost and programming which would be necessary with MDBS III.

APPENDIX A

CP/M SYSTEM FUNCTION CALLS

SYSTEM RESET
CONSOLE INPUT
CONSOLE OUTPUT
READER INPUT
PUNCH OUTPUT
LIST OUTPUT
DIRECT CONSOLE I/O
GET I/O BYTE
SET I/O BYTE
PRINT STRING
READ CONSOLE BUFFER
GET CONSOLE STATUS
RETURN CP/M VERSION NUMBER
RESET DISK SYSTEM
SELECT DISK
OPEN FILE
CLOSE FILE
SEARCH FOR FIRST FILE
SEARCH FOR NEXT FILE
DELETE FILE
READ FILE SEQUENTIAL
WRITE FILE SEQUENTIAL
MAKE FILE
RENAME FILE
RETURN LOGIN VECTOR
RETURN CURRENT DISK
SET DMA ADDRESS
GET ADDRESS OF DISK ALLOCATION VECTOR
WRITE PROTECT DISK
GET READ ONLY VECTOR
SET FILE ATTRIBUTES
GET ADDRESS OF DISK PARAMETERS
SET/GET USER CODE NUMBER
READ FILE RANDOM
WRITE FILE RANDOM
COMPUTE FILE SIZE
SET RANDOM RECORD
RESET DRIVE
WRITE RANDOM WITH ZERO FILL

APPENDIX B

MP/M SYSTEM FUNCTION CALLS

SYSTEM RESET
CONSOLE INPUT
CONSOLE OUTPUT
READER INPUT
PUNCH OUTPUT
LIST OUTPUT
DIRECT CONSOLE I/O
GET I/O BYTE
SET I/O BYTE
PRINT STRING
READ CONSOLE BUFFER
GET CONSOLE STATUS
RETURN CP/M VERSION NUMBER
RESET DISK SYSTEM
SELECT DISK
OPEN FILE
CLOSE FILE
SEARCH FOR FIRST FILE
SEARCH FOR NEXT FILE
DELETE FILE
READ FILE SEQUENTIAL
WRITE FILE SEQUENTIAL
MAKE FILE
RENAME FILE
RETURN LOGIN VECTOR
RETURN CURRENT DISK
SET DMA ADDRESS
GET ADDRESS OF DISK ALLOCATION VECTOR
WRITE PROTECT DISK
GET READ ONLY VECTOR
SET FILE ATTRIBUTES
GET ADDRESS OF DISK PARAMETERS
SET/GET USER CODE NUMBER
READ FILE RANDOM
WRITE FILE RANDOM
COMPUTE FILE SIZE
SET RANDOM RECORD
RESET DRIVE
ACCESS DRIVE
FREE DRIVE
WRITE RANDOM WITH ZERO FILL
TEST AND WRITE RECORD
LOCK RECORD
UNLOCK RECORD
SET MULTI-SECTOR RECORD
SET BDOS ERROR COUNT
GET DISK FREE SPACE
CHAIN TO PROGRAM

FLUSH DISK BUFFERS
SET DIRECTORY LABEL
READ FILE EXTENDED FILE CONTROL BLOCK
WRITE FILE EXTENDED FILE CONTROL BLOCK
SET DATE AND TIME
GET DATE AND TIME
SET DEFAULT PASSWORD
RETURN MP/M SERIAL NUMBER
ABSOLUTE MEMORY REQUEST
RELOCATABLE MEMORY REQUEST
MEMORY FREE
POLL
FLAG WAIT
FLAG SET
MAKE QUEUE
OPEN QUEUE
DELETE QUEUE
READ QUEUE
CONDITIONAL READ QUEUE
WRITE QUEUE
CONDITIONAL WRITE QUEUE
DELAY
DISPATCH
TERMINATE PROCESS
CREATE PROCESS
SET PRIORITY
ATTACH CONSOLE
DETACH CONSOLE
SET CONSOLE
ASSIGN CONSOLE
SEND COMMAND LINE TO INTERPRETER
CALL RESIDENT SYSTEM PROCESS
PARSE FILENAME
GET CONSOLE NUMBER
SYSTEM DATA ADDRESS
GET DATE AND TIME
RETURN PROCESS DESCRIPTOR ADDRESS
ABORT SPECIFIED PROCESS
ATTACH LIST DEVICE
DETACH LIST DEVICE
SET LIST DEVICE
CONDITIONAL ATTACH LIST DEVICE
CONDITIONAL ATTACH CONSOLE
RETURN MP/M VERSION NUMBER
GET LIST DEVICE NUMBER

APPENDIX C

CP/NET SYSTEM FUNCTION CALLS

SYSTEM RESET
CONSOLE INPUT
CONSOLE OUTPUT
READER INPUT
PUNCH OUTPUT
LIST OUTPUT
DIRECT CONSOLE I/O
GET I/O BYTE
SET I/O BYTE
PRINT STRING
READ CONSOLE BUFFER
GET CONSOLE STATUS
RETURN CP/M VERSION NUMBER
RESET DISK SYSTEM
SELECT DISK
OPEN FILE
CLOSE FILE
SEARCH FOR FIRST FILE
SEARCH FOR NEXT FILE
DELETE FILE
READ FILE SEQUENTIAL
WRITE FILE SEQUENTIAL
MAKE FILE
RENAME FILE
RETURN LOGIN VECTOR
RETURN CURRENT DISK
SET DMA ADDRESS
GET ADDRESS OF DISK ALLOCATION VECTOR
WRITE PROTECT DISK
GET READ ONLY VECTOR
SET FILE ATTRIBUTES
GET ADDRESS OF DISK PARAMETERS
SET/GET USER CODE NUMBER
READ FILE RANDOM
WRITE FILE RANDOM
COMPUTE FILE SIZE
SET RANDOM RECORD
RESET DRIVE
WRITE RANDOM WITH ZERO FILL
LOGIN NETWORK
LOGOFF NETWORK
SEND MESSAGE ON NETWORK
RECEIVE MESSAGE FROM NETWORK
GET NETWORK STATUS
GET CONFIGURATION TABLE ADDRESS

APPENDIX D

MDBS III COMMANDS

* MDBS III DML COMMANDS *

ASSIGNMENT COMMANDS

SET CURRENT OF RUN UNIT BASED ON MEMBER
SET CURRENT OF RUN UNIT TO NULL
SET CURRENT OF RUN UNIT BASED ON OWNER
SET CURRENT OF RUN UNIT BASED ON USER INDICATOR
SET MEMBER BASED ON CURRENT OF RUN UNIT
SET MEMBER TO CURRENT OF RUN UNIT
SET MEMBER BASED ON MEMBER
SET MEMBER TO NULL
SET MEMBER BASED ON OWNER
SET MEMBER BASED ON USER INDICATOR
SET OWNER BASED ON CURRENT OF RUN UNIT
SET OWNER TO CURRENT OF RUN UNIT
SET OWNER BASED ON MEMBER
SET OWNER TO NULL
SET OWNER BASED ON OWNER
SET OWNER BASED ON USER INDICATOR
SET USER INDICATOR TO CURRENT OF RUN UNIT
SET USER INDICATOR TO MEMBER
SET USER INDICATOR TO NULL
SET USER INDICATOR TO OWNER
SET USER INDICATOR TO USER INDICATOR

BOOLEAN COMMANDS

AND OF MEMBERS WITH MEMBERS
AND OF MEMBERS WITH OWNERS
AND OF OWNERS WITH MEMBERS
AND OF OWNERS WITH OWNERS
EXCLUDE MEMBERS FROM MEMBERS
EXCLUDE MEMBERS FROM OWNERS
EXCLUDE OWNERS FORM MEMBERS
EXCLUDE OWNERS FROM OWNERS

CREATE/DELETE COMMANDS

CREATE RECORD IN AREA
CREATE RECORD AND STORE
DELETE RECORD THAT IS CURRENT
DELETE RECORD THAT IS MEMBER
DELETE RECORD THAT IS OWNER

CONNECT/DISCONNECT COMMANDS

INSERT MEMBER INTO SET
INSERT OWNER INTO SET
REMOVE MEMBER FROM SET
REMOVE OWNER FROM SET
REMOVE ALL SET MEMBERS
REMOVE ALL SET OWNERS

FIND COMMANDS

FIND DUPLICATE RECORD BASED ON CALC KEY
FIND FIRST MEMBER
FIND FIRST OWNER
FIND FIRST SEQUENTIAL RECORD
FIND LAST MEMBER
FIND LAST OWNER
FIND MEMBER BASED ON DATA ITEM
FIND MEMBER BASED ON SORT KEY
FIND NEXT MEMBER
FIND NEXT MEMBER BASED ON DATA ITEM
FIND NEXT MEMBER BASED ON SORT KEY
FIND NEXT OWNER
FIND NEXT OWNER BASED ON DATA ITEM
FIND NEXT OWNER BASED ON SORT KEY
FIND NEXT SEQUENTIAL RECORD
FIND OWNER BASED ON DATA ITEM
FIND OWNER BASED ON SORT KEY
FIND PRIOR MEMBER
FIND PRIOR OWNER
FIND RECORD BASED ON CALC KEY

MODIFY COMMANDS

PUT DATA INTO FIELD OF CURRENT OF RUN UNIT
PUT DATA INTO FIELD OF MEMBER
PUT DATA INTO FIELD OF OWNER
PUT DATA INTO CURRENT OF RUN UNIT
PUT DATA INTO MEMBER
PUT DATA INTO OWNER

RETRIEVAL COMMANDS

GET DATA FROM CURRENT OF RUN UNIT
GET DATA FROM MEMBER
GET DATA FROM OWNER
GET FIELD FROM CURRENT OF RUN UNIT
GET FIELD FROM MEMBER
GET FIELD FROM OWNER

MULTIUSER LOCKING COMMANDS

MULTI USER ACTIVE INDICATOR
MULTIUSER CONTENTION COUNT
MULTIUSER CURRENT OF RUN UNIT FREE
MULTIUSER CURRENT OF RUN UNIT PROTECT
MULTIUSER RECORD TYPE FREE
MULTIUSER RECORD TYPE PROTECT
MULTIUSER SET FREE
MULTIUSER SET PROTECT

RECOVERY COMMANDS

LOG START OF COMPLEX TRANSACTIONS
LOG END OF COMPLEX TRANSACTIONS
LOG FILE SPECIFICATION
LOG FILE BUFFER FLUSH
LOG FILE MESSAGE
PRE-IMAGE FILE DECLARATION
TRANSACTION ABORT
TRANSACTION BEGIN
TRANSACTION COMMIT

SPECIAL COMMANDS

ALTER END OF SET
DEFINE DATA BLOCK
EXTEND DATA BLOCK
SET PAGE BUFFER REGION
UNDEFINE DATA BLOCKS
VARIABLE FOR COMMAND STATUS

UTILITY COMMANDS

ALLOCATE USER INDICATORS
CHECK CURRENT OF RUN UNIT AGAINST USER INDICATOR
DATA BASE CLOSE
DATA BASE CLOSE FOR AREA
DATA BASE ENVIRONMENT
DATA BASE OPEN
DATA BASE OPEN AREA
DATA BASE SAVE
DATA BASE STATISTICS
GET MEMBER COUNT
GET OWNER COUNT
GET TYPE OF CURRENT OF RUN UNIT
GET TYPE OF MEMBER
GET TYPE OF OWNER
NULL ALL CURRENCY INDICATORS
TEST CURRENT OF RUN UNIT TYPE
TEST MEMBER TYPE
TEST OWNER TYPE

* MDBS III QRS COMMANDS *

DISPLAY
LIST
WRITE
SPEW
STATS
COMPUTE
SET
DEFINE
READ
OPEN
CLOSE
QUIT

APPENDIX E

DBASE II COMMANDS, FUNCTIONS AND OPERATORS

DATABASE STRUCTURE COMMANDS

CREATE DATABASE STRUCTURE
DISPLAY DATABASE STRUCTURE
MODIFY DATABASE STRUCTURE

FILE OPERATIONS

RENAME FILE
DISPLAY FILES ON DRIVE

DATABASE OPERATIONS

USE DATABASE
SELECT PRIMARY/SECONDARY DATABASE
RENAME DATABASE
COPY DATABASE
SORT DATABASE
INDEX DATABASE
UPDATE DATABASE
JOIN
PACK
APPEND FROM DATABASE
DISPLAY DATABASE
BROWSE

DATABASE RECORD OPERATIONS

DELETE RECORD
RECALL DELETED RECORD
EDIT RECORD
REPLACE FIELD
INSERT RECORD

MEMORY VARIABLE COMMANDS

DISPLAY MEMORY VARIABLES
STORE VALUE TO MEMORY VARIABLE
CANCEL MEMORY VARIABLES
SAVE MEMORY TO DISK FILE
RESTORE MEMORY FROM DISK FILE

INTERACTIVE INPUT COMMANDS

WAIT FOR INPUT
ACCEPT CHARACTER STRING INPUT
READ FORMATTED INPUT

DATABASE POSITIONING COMMANDS

SKIP
GOTO RECORD
FIND RECORD
LOCATE RECORD

MISC DBMS COMMANDS

QUIT
CLEAR
REPORT
SET

PROGRAMMING COMMANDS

DO COMMAND FILE
IF THEN ELSE
DO WHILE
DO CASE

DBASE II FUNCTIONS

BOOLEAN FUNCTIONS

IS THIS A DELETED RECORD
IS THIS RECORD END OF FILE
DOES FILE EXIST

CONVERSION FUNCTIONS

LOWER TO UPPERCASE
CONVERT REAL TO INTEGER
CONVERT STRING TO INTEGER
CONVERT INTEGER TO STRING
INTEGER TO ASCII

GENERAL FUNCTIONS

RETURN DATA TYPE
RETURN STRING LENGTH
RETURN SUBSTRING
SUBSTRING SEARCH
ELIMINATE TRAILING BLANKS FROM STRING
RETURN DATE

DBASE II OPERATORS

ARITHMETIC OPERATOR

ADD

SUBTRACT
MULTIPLY
DIVIDE

COMPARISON OPERATOR

LESS THAN
GREATER THAN
EQUAL
NOT EQUAL
LESS THAN OR EQUAL
GREATER THAN OR EQUAL

LOGICAL OPERATOR

AND
NOT
OR

STRING OPERATOR

CONCATENATE
CONCATENATE WITH BLANK REMOVAL

APPENDIX F

LIMITATIONS AND CONSTRAINTS

DBASE II

Number of fields per record	32
Number of characters per record	1000
Number of records per database	65535
Number of characters per character string	254
Accuracy of numeric fields	10 digits
Number of variables	64
Number of characters per command line	254
Number of files open at one time	16

MDBS III with PL/I-80

Number of fields per record	limited by compiler
Number of characters per record	limited by compiler
Number of records per database	limited by storage
Number of characters per character string	254
Accuracy of numeric fields	15 digits
Number of variables	limited by compiler
Number of characters per command line	255
Areas per database	16
Record types per database	255
Sets per database	none
Owner record occurrences	none
Member record occurrence	none
Owner record types	127
Member record types per set	127

APPENDIX G

GENERAL FEATURES COMPARISON

FEATURE	DBASE II	MDBS III
Data Model	relational	network
Multiple Files	yes	yes
indexing	yes	yes
sorted insertion	no	yes
host language	no	yes
interactive queries	yes	yes
concurrency control	no	yes
range constraints	no	yes
security	none	access list
crash recovery	no	yes
memory requirement	48K	64K
storage transparent to user	no	yes
user designed full screen I/O	yes	no
batch commands available	yes	yes
user should be programmer	no	yes
max number of fields	32	host language
audit trail produced	no	yes
predefined updates	yes	yes
predefined queries	yes	yes
report capabilities	yes	yes
summary breakpoints in reports	yes	yes
statistics from queries	no	yes

APPENDIX H

PERFORMANCE TEST PROBLEM DESCRIPTIONS

PROBLEMS FOR PHYSICS DEPARTMENT EQUIPMENT

1. Using MDBS III QRS and a DBASE II command file, locate and list all equipment manufactured by Triplett and Simpson. Also count and display the total number of records which meet this criteria. Create a report which lists all this data and has appropriate column headings. The report should include the following fields: item name, model number, serial number,cost and location.

2. Create a database with 20 records containing the following fields: model number, manufacturer, calibration frequency, person responsible for calibration. The model number and manufacturer are the same as fields 3 and 6 in the original database. The rest of the fields are created using made up names and calibration frequencies. Use the two database to create a file in SDF form which lists model number, manufacturer, item number, location, calibration frequency and person responsible for calibration.

3. Using the original database, create a text file in SDF form. This file will contain the following fields which are a subset of the original database: item name, model number, manufacturer and cost. Total the cost and place it at the bottom of the file.

4. Using the original database, create a duplicate database which is sorted by fund code.

PROBLEMS FOR ENGINEERING CURRICULUM STUDENTS

1. Using MDBS III QRS and a DBASE II command file, locate and list all students who are not citizens of the United States. Count and display the number of students who meet this criteria. Create a report which lists these records with appropriate column headings. This report should contain the following fields: name, rank, service, section, graduation date and curriculum.

2. Create a second database with 20 records containing the following fields: monthly pay, basic allowance, rank and country. Rank and country are the same as the fields in the original database and the rest are made up. Use the two databases to create a file in SDF form which has the following fields: name, rank, country, monthly pay, basic allowance and section.

3. Using the original database, create a text file in SDF form. This file will contain the following fields which are a subset of the original database: name, rank, section, curriculum and address.

4. Using the original database, create a duplicate database sorted by rank.

PROBLEMS FOR LIBRARY PERIODICALS

1. Using MDBS III QRS and DBASE II command files, locate and list all entries which are current and are published quarterly. Also count and display the number of entries which meet this criteria. Create a report which list these records with appropriate column headings. This report should contain the following fields: reference number, retention time, account number and binding type.

2. Create a second database with 20 records containing the following fields: account number, person responsible and office location. The account number is the same as field 9 in the original database. The rest of the fields are made up. Use the two databases to create a file in SDF form which lists account number, person responsible, office location, reference number, retention period.

3. Using the original database, create a text file in SDF form. This file will contain the following fields which are a subset of the original database: reference number, binding type, binding date and account number.

4. Using the database originally created, create a duplicate database sorted by account number.

NOTE: The term SDF refers to a text file with fixed length records similar to those used in COBOL. Each record consists of a string of characters terminated with a carriage return and line feed. The fields are located in fixed columns within the character string.

APPENDIX I

TIME PERFORMANCE TABLE
(records/minute)

	PROBLEM 1	PROBLEM 2	PROBLEM 3	PROBLEM 4
DBASE II	396	48	246	66
MDBS III	240	18	180	42

BIBLIOGRAPHY

Date, C. J., An Introduction to Database Systems, Addison-Wesley Publishing Company, 1977.

Ullman, J. D., Principles of Database Systems, Computer Science Press, 1980.

Ashton-Tate, DBASE II Reference Guide, 1980.

Digital Research, CP/M Interface Guide, 1980.

Digital Research, CP/NET User's Guide, 1980.

Digital Research, MP/M User's Guide, 1980.

Digital Research, MP/M II Programmer's Guide, 1981.

Micro Data Base Systems, MDBS Application Programming Reference Manual, 1981.

Micro Data Base Systems, MDBS Data Base Design Reference Manual, 1981.

Micro Data Base Systems, MDBS Query Retrieval Manual, 1981.

Micro Data Base Systems, MDBS System Specific Installation Manual, 1981.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
3.	Department Chairman, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93940	1
4.	Dushan Badal, Code 52 Department of Computer Science Naval Postgraduate School Monterey, California 93940	5
5.	LCDR Richard Relue, USN 505 East Second St. Defiance, Ohio 43512	1

LME

-83